JAVA COLLECTIONS FRAMEWORK

Prior to Java 2, Java provided ad hoc classes such as **Dictionary, Vector, Stack**, and **Properties** to store and manipulate groups of objects. Although these classes were quite useful, they lacked a central, unifying theme. Thus, the way that you used Vector was different from the way that you used Properties.

The collections framework was designed to meet several goals.

- The framework had to be high-performance. The implementations for the fundamental collections (dynamic arrays, linked lists, trees, and hash tables) are highly efficient.
- The framework had to allow different types of collections to work in a similar manner and with a high degree of interoperability.
- Extending and/or adapting a collection had to be easy.

Toward this end, the entire collections framework is designed around a set of standard interfaces. Several standard implementations such as **LinkedList, HashSet,** and **TreeSet**, of these interfaces are provided that you may use as-is and you may also implement your own collection, if you choose.

A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- **Implementations i.e. Classes:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface.

In addition to collections, the framework defines several map interfaces and classes. Maps store key/value pairs. Although maps are not *collections* in the proper use of the term, but they are fully integrated with collections.

The Collection Interfaces:

The collections framework defines several interfaces. This section provides an overview of each interface:

SN	Interfaces with Description
1	The Collection Interface This enables you to work with groups of objects; it is at the top of the collections hierarchy.
2	The List Interface This extends **Collection** and an instance of List stores an ordered collection of elements.
3	The Set This extends Collection to handle sets, which must contain unique elements

4	The SortedSet This extends Set to handle sorted sets
5	The Map This maps unique keys to values.
6	The Map.Entry This describes an element (a key/value pair) in a map. This is an inner class of Map.
7	The SortedMap This extends Map so that the keys are maintained in ascending order.
8	The Enumeration This is legacy interface and defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects. This legacy interface has been superceded by Iterator.

The Collection Classes:

Java provides a set of standard collection classes that implement Collection interfaces. Some of the classes provide full implementations that can be used as-is and others are abstract class, providing skeletal implementations that are used as starting points for creating concrete collections.

The standard collection classes are summarized in the following table:

SN	Classes with Description
1	**AbstractCollection** Implements most of the Collection interface.
2	**AbstractList** Extends AbstractCollection and implements most of the List interface.
3	**AbstractSequentialList** Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
4	LinkedList Implements a linked list by extending AbstractSequentialList.
5	ArrayList Implements a dynamic array by extending AbstractList.
6	**AbstractSet** Extends AbstractCollection and implements most of the Set interface.
7	HashSet Extends AbstractSet for use with a hash table.
8	LinkedHashSet Extends HashSet to allow insertion-order iterations.
9	TreeSet Implements a set stored in a tree. Extends AbstractSet.
10	**AbstractMap**

	Implements most of the Map interface.
11	HashMap Extends AbstractMap to use a hash table.
12	TreeMap Extends AbstractMap to use a tree.
13	WeakHashMap Extends AbstractMap to use a hash table with weak keys.
14	LinkedHashMap Extends HashMap to allow insertion-order iterations.
15	IdentityHashMap Extends AbstractMap and uses reference equality when comparing documents.

The *AbstractCollection, AbstractSet, AbstractList, AbstractSequentialList* and *AbstractMap* classes provide skeletal implementations of the core collection interfaces, to minimize the effort required to implement them.

The following legacy classes defined by java.util has been discussed in previous tutorial:

SN	Classes with Description
1	Vector This implements a dynamic array. It is similar to ArrayList, but with some differences.
2	Stack Stack is a subclass of Vector that implements a standard last-in, first-out stack.
3	Dictionary Dictionary is an abstract class that represents a key/value storage repository and operates much like Map.
4	Hashtable Hashtable was part of the original java.util and is a concrete implementation of a Dictionary.
5	Properties Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String.
6	BitSet A BitSet class creates a special type of array that holds bit values. This array can increase in size as needed.

The Collection Algorithms:

The collections framework defines several algorithms that can be applied to collections and maps. These algorithms are defined as static methods within the Collections class.

Several of the methods can throw a **ClassCastException**, which occurs when an attempt is made to compare incompatible types, or an **UnsupportedOperationException**, which occurs when an attempt is made to modify an unmodifiable collection.

Collections defines three static variables: EMPTY_SET, EMPTY_LIST, and EMPTY_MAP. All are immutable.

SN	Algorithms with Description
1	The Collection Algorithms Here is a list of all the algorithm implementation.

How to use an Iterator ?

Often, you will want to cycle through the elements in a collection. For example, you might want to display each element.

The easiest way to do this is to employ an iterator, which is an object that implements either the Iterator or the ListIterator interface.

Iterator enables you to cycle through a collection, obtaining or removing elements. ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.

SN	Iterator Methods with Description
1	Using Java Iterator Here is a list of all the methods with examples provided by Iterator and ListIterator interfaces.

How to use an Comparator ?

Both TreeSet and TreeMap store elements in sorted order. However, it is the comparator that defines precisely what *sorted order* means.

This interface lets us sort a given collection any number of different ways. Also this interface can be used to sort any instances of any class.(even classes we cannot modify).

SN	Iterator Methods with Description
1	Using Java Comparator Here is a list of all the methods with examples provided by Comparator Interface.

Summary:

The Java collections framework gives the programmer access to prepackaged data structures as well as to algorithms for manipulating them.

A collection is an object that can hold references to other objects. The collection interfaces declare the operations that can be performed on each type of collection.

The classes and interfaces of the collections framework are in package java.util.